Graph Processing For Spatial Network Queries

Kevin Shaw, John Sample, Elias Ioup
Naval Research Laboratory
Stennis Space Center
Mississippi
{shaw, jsample, eioup}@nrlssc.navy.mil
Tel: (228) 688-4197

Olivier Mansion
Department of Computer Science
Ecole Centrale de Nantes
France
olivier.mansion@eleves.ec-nantes.fr

Mahdi Abdelguerfi
Computer Science Department
University of New Orleans
Louisiana
mahdi@uno.edu
Tel: (985) 626-9432

ABSTRACT

Spatial network queries often have performance bound by the structure and size of the underlying network. This paper discusses methods of improving the performance of these queries, specifically those using network expansion algorithms, by creating a graph representation of the network and removing unnecessary nodes and edges. In addition, two methods of storing and accessing the graph are compared for speed and usefulness in different applications. A main memory approach to graph storage using a shared library is compared to a database storage approach. Though fast, the main memory approach has some limitations in its usefulness.

Keywords

spatial databases, network expansion, KNN query, range query.

1. INTRODUCTION

Work on spatial databases has been extensive over the years. However, the work has for the most part been limited to systems where objects may be located anywhere on the Euclidean plane. Just as important are systems where objects are limited to locations on a network such as roads or rivers. The problems and solutions involved in these systems are very different from Euclidean systems. Managing large spatial network databases requires developing separate solutions.

Papadias, *et. al*. [1] created a comprehensive approach to querying spatial network databases. They introduced network expansion as an efficient technique for performing nearest neighbor (KNN) and range queries over objects constrained to a network. Network expansion remains a widely used technique for performing these queries and is the benchmark by which other techniques are measured [2]. The algorithm is a simple, yet efficient, method of performing the queries.

The network expansion, and any other network query algorithm, requires a network to operate upon. Usually, this network takes the form of a graph. This graph usually must be created from some underlying network dataset. Some preprocessing work must be done simply to create an efficient, usable graph. More preprocessing may even increase the efficiency of the application.

This paper discusses efficient methods of creating and storing the spatial network to increase the performance of network expansion and other spatial network algorithms. The network is initially created and stored as a graph in a database. Unnecessary nodes are removed from the graph to decrease storage requirements and query execution times. The entire process is performed in parallel on a Beowulf cluster which reduces the computation time substantially.

Also discussed are two different methods of storing the network. One method uses a shared library which keeps the graph in main memory. This method is fast but is not scalable and places great demand upon the underlying system. Using a database for graph storage is much slower but is also much more scalable

2. BACKGROUND

2.1 Spatial Network Queries

Two types of queries are used commonly in spatial networks: range queries and nearest neighbor (KNN) queries. A range query finds all objects whose distance is within a certain range from a given query point. Below is a formal definition of a range query R:

$$R = \{o \in M \mid d(p,o) < r\}$$

M = points of interest

p = query point

$d(x,y)$ = distance over network from point x to y

r = max distance from the query point

The KNN query returns the K closest objects to the query point. KNN is the definition of set returned by a nearest neighbor query:

$$KNN = \{N \subset M \mid |N| = k \wedge \forall o \notin N, q \in N \Rightarrow d(p,o) > d(p,q)\}$$

K = number of objects to return

2.2 Network Expansion

Papadias, *et. al*. [1] suggest two general methods for performing KNN and range queries. The first is Euclidean restriction. This method exploits the fact that the network distance of a point is always greater than or equal to the Euclidean distance. The other method is network expansion. Network expansion is the faster algorithm and is used in other spatial network queries, such as the moving object KNN query by Jensen, *et. al*. [3].

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
14 OCT 2005	**Conference Proceedings, (refereed)**	**14-10-2005 to 26-06-2006**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Graph Processing For Spatial Network Queries	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Kevin Shaw; John Sample; Elias Ioup; Olivier Mansion; Mahdi Abdelguerfi	5e. TASK NUMBER
	5f. WORK UNIT NUMBER **74810005**
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Marine Geosciences Division,Stennis Space Center,MS,39529-5004**	8. PERFORMING ORGANIZATION REPORT NUMBER **NRL/PP/7440--05-1017**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) **Naval Research Laboratory, 1005 Balch Blvd., Stennis Space Center, MS, 39529**	10. SPONSOR/MONITOR'S ACRONYM(S) **NRL**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Spatial network queries often have performance bound by the structure and size of the underlying network. This paper discusses methods of improving the performance of these queries, specifically those using network expansion algorithms, by creating a graph representation of the network and removing unnecessary nodes and edges. In addition, two methods of storing and accessing the graph are compared for speed and usefulness in different applications. A main memory approach to graph storage using a shared library is compared to a database storage approach. Though fast, the main memory approach has some limitations in its usefulness.

15. SUBJECT TERMS
spatial databases, network expansion, KNN query, range query

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**6**	

It is also the benchmark used to compare to other spatial network query algorithms, such as the Voronoi based KNN query introduced by Kolahdouzan, and Shahabi [2].

Network expansion works by traversing the network from the initial query point, similar to Dijkstra's algorithm. As it traverses the network it adds points of interest that match the query as it reaches them. For KNN queries the algorithm continues until K points have been retrieved. For range queries the algorithm continues until all segments within the given range are traversed. Like Dijkstra's algorithm, network expansion has complexity dependent on the number of nodes and edges in the network.

2.3 Network Model

A real world spatial network such as a system of roads must be modeled for use in a computer system. The modeled network will be in two dimensions instead of three and the network is approximated as lines. Nodes are placed at intersections and endpoints in the network. The piece of a network between two nodes is a segment. Segments are the basic type of the network.

Each segment is defined as a polyline. A polyline L is a sequence of points:

$$L \equiv (p_1, p_2, \ldots, p_n) \text{ where } p_i \in \Re^2$$

A node N in the network is represented as:

$$N \equiv p_i \in \Re^2 \text{ where } p_i = p_1 \in L = (p_1, \ldots, p_n) \text{ or}$$
$$p_i = p_n \in L = (p_1, \ldots, p_n)$$

Nodes indicate all the connections between road segments and therefore indicate how travel can take place in the network. If two road segments do not share a node then they do not directly connect.

In this model segments are bidirectional. The assumption of bidirectional segments is not important to methods described later and small modifications will allow support for unidirectional segments

The spatial network may be modeled as a graph as done by Jensen, *et. al*. [3]. The graph representation simplifies the network by removing the geometry. Because the network expansion algorithm functions similarly to classical graph search algorithms representing the network as a graph poses no difficulty. What follows is a discussion of how to create and process the graph to increase the efficiency of network expansion as well as other spatial network algorithms.

3. GRAPH PROCESSING

3.1 The Data Set

The actual data for the spatial network may come in many different forms. For this research, TIGER data from the US Census was used as the underlying network. The primary reason for using this specific data is that it is free to the public and provides the road network for the entire United States. For these experiments only two states were used, Louisiana and California.

Another possible source of spatial network data is provided by companies such as Navtech and TeleAtlas. The data provided by these companies is more accurate and better formatted for use in actual spatial network applications. However, these data sets are expensive and thus not available for this work. The methods described in this work are applicable to these commercial data sets and other spatial networks which are not roads.

3.2 Graph Creation

The initial network data is organized as network segments which are encoded as a sequence of latitude and longitude coordinates. First, the two endpoints of each network segment are created as nodes in the network graph. In this system, the points of interest are also included as nodes in the graph. Including points of interest as nodes in the graph is not necessary to the graph processing described below. Often the points of interest are too dynamic a set to hard code them into the graph. If points of interest are not included as nodes, it is required that they be explicitly linked to the network segment on which they lie.

The graph is stored as a sequence of adjacency lists, one for each node. The following information is stored in the graph for each node in the network:

1. The id of the node.
2. The coordinates of the node.
3. The number of adjacent nodes.
4. An array of references to adjacent nodes.
5. The distance to each adjacent node.
6. A reference to point of interest information if this node is a point of interest.

The above fields must be updated as the graph is processed. Below are the important changes that are made to the graph of the network.

The original network data does not necessarily ensure correct connectivity between network segments. Two segments which are connected in the real world may have endpoints whose coordinates are slightly different. These nodes should have the same coordinates and be labeled as the same node. All nodes should be compared to ensure that small decimal errors do not differentiate nodes which should be the same. An R-Tree over the node coordinates will allow efficient search for nodes which should be merged. It is essential that the network segments be updated with new node equivalences to ensure that connectivity information is preserved.

After all the nodes are determined the graph is pruned of unessential nodes and segments. Nodes and line segments which contain points of interest will not be pruned. These sections of the graph contain the information important to the network queries an must stay intact. Jensen, *et. al*. [3] proposed such a technique and removed all nodes with two adjacencies. Removing nodes with only two adjacencies is justifiable

because these nodes do not model connectivity to an alternate path. Collapsing these segments into one segment cannot alter possible paths taken by a graph traversal algorithm.

Nodes with one adjacency may also be removed from the graph. It is these nodes which often allow the greatest amount of pruning to be performed. These nodes represent end sections of the graph which contain no useful information to answer queries and no connectivity information. Figures 1 through 3 show an example of nodes pruned from a graph. Yellow nodes are regular graph nodes. The red nodes are actually points of interest and are exempt from the pruning requirements. In Figure 1, the nodes with one adjacency are removed from the network. Figure 2 shows nodes with two adjacencies pruned. The final network with all nodes removed is shown in Figure 3.

The resultant graph after pruning will have fewer nodes and line segments. Most queries on the pruned graph will run faster because their complexity is based on the number of nodes or line segments in the graph.

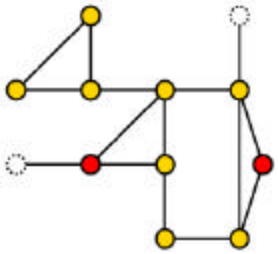

Figure 1. Remove nodes with one adjacency.

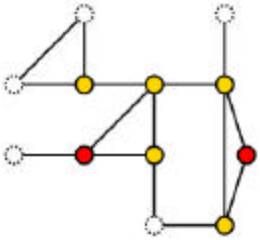

Figure 2. Remove nodes with two adjacencies.

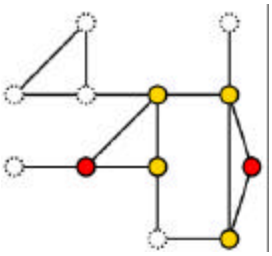

Figure 3. All nodes pruned from network.

The process of removing nodes is an iterative one. Node are repeatedly removed from the graph until no remaining nodes fit the pruning categories. The effectiveness of removing these extra nodes depends on the original network data. Some network data contains many of these extraneous nodes and segments whereas other data are organized to not have the extra line segments. Pruning will be more effective in the former rather than the latter. Most network data will be reduced in size by the pruning, especially the removal of end nodes. Figure 4 shows an area of roads in Louisiana with the resulting pruned graph overlaid. Notice the decrease in complexity of the pruned graph.

Figure 4. Example of graph reduction.

3.3 Parallelizing the Process

Because removal of nodes is an iterative process, it will take quite a while to complete on a large and complex network. If possible, running the graph creation in parallel on a cluster will reduce the required time substantially. The pruning process parallelizes easily. All slaves are given a copy of the network database. The network is split into disjoint geographic boxes by the master and assigned to each of the slaves. The slave removes the unnecessary nodes from its area and sends the changes back to the master. Once all slaves have finished the current iteration, new work is assigned and the process repeats itself until all work is finished. Pseudo code for the above algorithm is shown in Figure 5.

```
BEGIN
| nslaves ?  number of slaves
| FOR i=1 To nslaves STEP 1 DO
| | work ?  next set of id to process
| | send work to the slave i
| END FOR
| work ?  next set of id to process
| WHILE work DO
| | wait for the signal work done of a slave
| | send work to the slave that just finished
| | work ?  next set of id to process
| END WHILE
| FOR i=1 To nslaves STEP 1 DO
| | wait for the signal work done of a slave
| END FOR
END
```

Figure 5. Algorithm for parallel graph reduction.

Two standard problems when parallelizing a process are how to handle the boundaries between slave areas and how to limit inter-process communication. Nodes whose adjacencies are in other slave areas will require special handling. And graph changes during the pruning process necessitate costly communication between the master and the slaves.

The solution to the boundary problem is to ignore all nodes whose adjacencies are not in the same area. These nodes are left to be handled by later iterations. After all slaves have processed their first iteration, the master must redistribute the work. By changing the location of the boundaries, the probability that a node will be near a boundary for each iteration is greatly reduced. If any boundary nodes do remain unhandled at the end of the pruning, the master node will make one final pass over the entire area and prune the problem nodes. An example of two slave areas is shown in Figure 6. The nodes which are removed have adjacencies only in the same slave area.

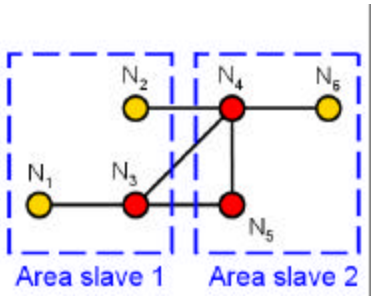

Figure 6. Two adjacent slave graph areas. Node N_2 is not pruned because its adjacency is in another slave area. N_1/N_6 is pruned by slave1/slave2.

To reduce communication within the cluster database updates are made selectively between the master and slaves. Slaves only send back necessary changes to the master and the master only updates the necessary portions of the slave database. For example, if a slave removes four nodes in the current iteration, only the information for those nodes need be sent back to the master. And when assigning work to a slave, only the area assigned should be updated in the slave's database. Other areas are inconsequential and should be left unchanged.

3.4 Graph Validation

For complex networks it will be difficult to ensure that the resulting graph is valid. A validation program checks to ensure correctness in the graph. The following are properties that should hold in the final graph:

- If A is adjacent to B then B is adjacent to A.
- If A is in the graph then all nodes adjacent to A are in the graph.
- $d(A,B) = d(B,A)$

4. GRAPH STORAGE AND ACCESS

This section compares two methods of storing and accessing the graph created above. Each method has its benefits and should be chosen with knowledge about the network data as well as the application requiring access to the graph.

4.1 Database Storage and Access

The first method of graph storage is to use a database. The database provides a simple method of managing the graph data. This method is highly scalable which is important when using large network data sets. Accessing the graph is done through basic SQL queries. The main concern with this method is how to program the spatial network queries. The test database used in this project is PostgreSQL [4], but the techniques apply to most database management systems (DBMS).

Two techniques are possible to access the data from a program implementing the spatial network queries. The first is to have a program external to the database perform the queries. The benefit of this method is that it is usually easy to write such a program. The choice of programming languages and tools is wide. Often, there are interfaces that allow the external program to be accessed from within the database and thus allow the queries to be used seamlessly. The problem with this method is that database access from the query program is usually slow. External connections to a database require overhead and limit performance, especially when there is a lot of data. Given that most useful network graphs are fairly large this is problematic.

The second method to access data from the network queries is to write an extension to the database. Instead of an external program, the queries are written as internal functions. As a result, the queries have the same access privileges and speed of native database functions. In PostgreSQL this functionality is provided by the Server Programming Interface (SPI).

SPI is simply a set of native interface functions that allow access to the Parser, Planner, Optimizer, and Executor of the DBMS. The benefit of SPI (or similar functions in other DBMS) is that user defined queries are performed fast. There is no connection overhead or inter-process communication. The downside is that using SPI is more complex that simply writing an external module. Language choice is limited when

extending the database functionality. SPI is written in C and full access to its functionality is only available using C (though some other languages provide limited accessibility). SPI also requires access to data using specific macros, functions, and data structures which can complicate the writing of functions. However, because SPI is the fastest method of accessing database data, it is our chosen method of interfacing with the database. Both KNN and range queries on the spatial network were implemented using the network expansion algorithm and the Server Programming Interface.

4.2 Shared Library Access

The second method of storing and accessing the data is through a dynamically accessible shared library. A special program generates a C source file containing the entire structure of the graph. All of the information contained in the database version of the graph is also contained in the shared library version. There are important benefits to this method but also some large deficiencies which may preclude its usefulness.

The shared library is actually a hard coded table of nodes. Each node is a global variable. The library also includes some statistics useful when using the graph like number of points of interest, etc. The graph itself is accessed through macros which simplify the interface and can be used directly through PostgreSQL.

The shared library is modifiable so that the network expansion algorithm can set flags on particular nodes. These flags indicate containment in particular lists used by the network expansion algorithm. In addition, the flags record the position of the node in a particular list. The result is that these lists rarely need to be scanned and the network expansion algorithm performs more efficiently. A similar fix could be done with the database access method but the list scan time costs much less than data retrieval and thus is not a large improvement.

The benefit of the shared library graph is speed. The graph is kept in main memory and results in extremely fast access times. Applications spatial network queries often require speedy results making the shared library a perfect access and storage method. Not only is access fast but loading the library is fast. Other main memory approaches would require loading the library from a data file or database. The data would need to be parsed and placed in the correct data structures. The shared library method does not require this startup overhead, making it perfect for an experimental environment with many tests and restarts. In particular, the shared library provides an excellent platform for comparing network expansion algorithms to other spatial network algorithms. The access speed benefit of the shared library approach is matched by other main memory approaches which may be more suited to an always on, deployed application.

The problems of the shared library graph, and other main memory approaches, are numerous however. The shared library is not a scalable method of storing the graph. A road network covering the entire United States would require about 2-3 gigabytes of main memory simply to hold the shared library. In addition, compiling the shared library requires even more memory. Another issue is that the shared library is static. Once changes are made to the graph the library requires a complete recomplication. Having to recreate a graph every time one node or segment changes is a significant problem.

These problems are not insurmountable. The memory usage is large but may be handled with current systems. Large networks may be stored in many shared libraries, possibly contained on many nodes in a cluster. The static nature of the shared library approach is not a problem with other main memory approaches which could be used with dynamic networks. In many cases, spatial networks are static data sets and there is no need to update the graph on regular intervals.

5. EXPERIMENTAL RESULTS

The data used in the experiments were the road networks of Louisiana and California. The road networks were obtained form the US Census Bureau Tiger/LINE data. The data was modified into the Shapefile file format and loaded into PostgreSQL using the PostGIS [5] extension. The cluster used in the experiments is a 72 node Beowulf cluster. It consists of 63 slave nodes which are 2.2Ghz Intel Pentium IV systems and 9 slave nodes which are 2.4Ghz Intel Pentium IV systems. All slave nodes have 1 GB of memory. The master node has dual 2.2GHz Intel Xeon processors and 2 GB of memory.

5.1 Graph Reduction

Table 1 indicates the initial statistics for the networks as well as the results of the graph reduction. The graph reduction is a useful step in the graph creation process for these datasets. The number of nodes in the Louisiana and California graphs are reduced by an order of magnitude. The reduction for these two networks are dramatic because of their initial structure, but most spatial networks will gain considerable benefit from the reduction step.

Table 1. Initial Statistics and Results of Graph Reduction

	Initial Nodes	Post-Reduction Nodes	Reduction Factor
Louisiana	1,200,091	137,947	88%
California	4,931,271	624,905	87%

5.2 Comparison of Access Methods

Each access method was tested using KNN queries and range queries over the Louisiana Network. Both the KNN query and the range query use the network expansion algorithms. The network expansion algorithms for the SPI and shared library access methods were kept similar; however, each algorithm was optimized for the particular access method used.

The results show that the shared library queries perform substantially faster than the SPI queries. Figure 7 and Figure 8 show the results for range and KNN queries plotted on a logarithmic scale. Figure 9 and Figure 10 show the results for small values of K and range plotted on a linear scale. The time

to perform KNN and range queries becomes asymptotic as K and range grow large because at a certain point the entire network is being expanded and there are no more paths to explore. Figure 11 and Figure 12 show the increase in speed of the shared library method in comparison to the SPI access method. Again the value becomes constant once the entire spatial network is expanded. An increase in the size of the network will show a greater speed improvement for larger values of K or range.

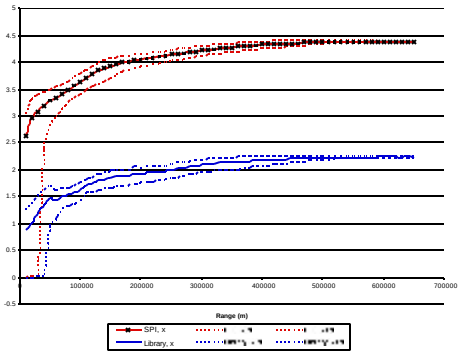

Figure 7. Range Query Comparison

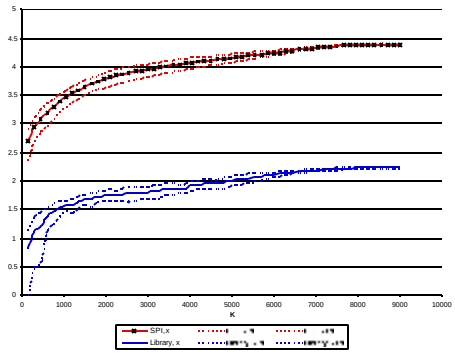

Figure 8. KNN Query Comparison

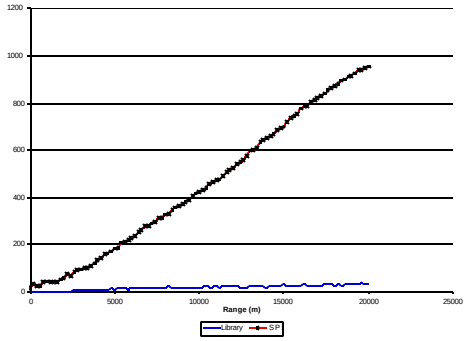

Figure 9. Range Query Comparison (Small K)

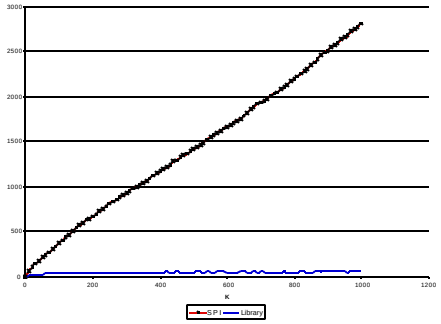

Figure 10. KNN Query Comparison (Small K)

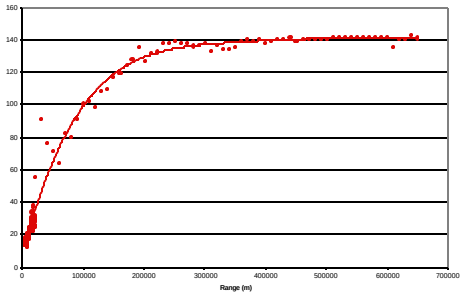

Figure 11. Increase in speed of shared library over SPI for range queries.

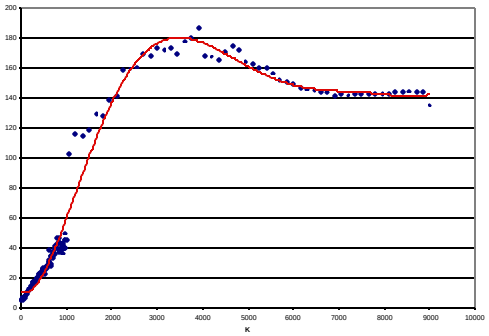

Figure 12. Increase in speed of shared library over SPI for KNN queries.

6. CONCLUSION

The graph processing described above is useful for a variety of applications. Spatial networks have become increasingly used by a variety of applications, including online mapping and in vehicle navigation systems. These systems always require high speed. By reducing the complexity of the network expansion and other network algorithms, graph reduction can provide a significant decrease in processing time.

The two graph storage and access methods discussed in this paper each have benefits. The shared library graph is an order of magnitude faster than the database stored graph and provides fast startup times for applications. However, applications have limited memory capacity and thus are unable to store the graph in main memory. With large networks the shared library approach becomes untenable without splitting the network into smaller pieces. Even though the database access is slow it is also highly scalable with respect to network size. With proper indexing, database access may provide performance ample for many applications.

7. REFERENCES

[1] D. Papdias, J. Zhang, N. Mamoulis, Y. Tao, *Query Processing in Spatial Network Databases*. Proceedings of the 29th VLDB Conference, 2003.

[2] M. Kolahdouzan, C. Shahabi. Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. Proceedings of the 30th VLDB Conference, pp. 804-851, 2004.

[3] C. Jensen, J. Kolar, T. Pedersen, I. Timko. Nearest Neighbor Queries in Road Networks. Proceedings of ACM GIS, pp. 1-8, 2003.

[4] PostgreSQL On-line Documentation: <http://www.postgresql.org/docs/>

[5] PostGIS Manual: <http://postgis.refractions.net/documentation/>